Optimal Scheduling for Real-Time Jobs in Energy Harvesting Computing Systems

Maryline Chetto

Optimal Scheduling for Real-Time Jobs in Energy Harvesting Computing Systems

Maryline Chetto,

Abstract—In this paper, we study a scheduling problem, in which every job is associated with a release time, deadline, required computation time and in addition required energy. We focus on an important special case where the jobs execute on a uniprocessor system that is supplied by a renewable energy source and uses a rechargeable storage unit with limited capacity. Earliest Deadline First (EDF) is a class one online algorithm in the classical real-time scheduling theory where energy constraints are not considered. We propose a semi-online EDF-based scheduling algorithm theoretically optimal (i.e. processing and energy costs neglected). This algorithm relies on the notions of energy demand and slack energy which are different from the well known notions of processor demand and slack time. We provide an exact feasibility test. There are no restrictions on this new scheduler: each job can be one instance of a periodic, aperiodic or sporadic task with deadline.

Index Terms—Real-time systems, energy harvesting, uniprocessor, optimal scheduling, earliest deadline first, slack energy.

1 Introduction

Energy harvesting is a technology that allows to capture otherwise unused ambient energy and convert it into electrical energy which can be used immediately or later through a storage unit [15], [27]. Ambient energy also known as environmental energy is obtained from natural and human-made sources that surround us in the environment (e.g. kinetic energy produced by movements). This approach extends the life of batteries (or eliminates them entirely) and decreases maintenance. A variety of techniques are available for energy harvesting, including solar, piezoelectricity, thermoelectricity, and physical motions. Energy harvesting appears to be a perfect match for wireless devices that otherwise rely on battery power. Some of the main applications include self-powered sensors in medical implants for health monitoring and embedded sensors in structures such as bridges and buildings for remote monitoring. The compactness -surface area and weight- that the device authorizes limits the yield of power that mostly hovers at milliwatt.

Low power design of electronic systems came in the focus of interest in the middle nineties. Power management techniques have been proposed to achieve energy efficiency of battery-powered devices. DPM (Dynamic Power Management) [24] and DVFS (Dynamic Voltage and Frequency Selection) [33] are the two conventional techniques that aim to reduce the static respectively dynamic energy dissipation. Nonetheless, these techniques alone do not prevent a battery from being replaced so that the device continues to operate. Nowadays, a growing number of applications involve many wireless sensors that may be deployed in wide areas and possibly unattainable places. Such systems should be designed to function perpetually without any human intervention because either costly or impractical. As a consequence, energy harvesting technology has been an area of rapid development during the last decade [14]. The introduction of energy harvesting capabilities into embedded systems such as wireless sensor networks introduces a lot of design questions. Firstly, how to intelligently use harvesting abilities so as to optimize its performance and lifetime? In other words, how to dynamically adapt the processing activity so as to subsist perpetually on a given energy source? Secondly, how to dimension the energy storage unit (e.g. battery or capacitor) and the harvester (e.g. solar panel) to guarantee an acceptable performance under all environmental conditions? Researchers also strive to design efficient power management techniques which additionally adapt to real-time requirements that characterize a lot of energy harvesting computing systems [28].

• *M. Chetto is with the University of Nantes, IRCCyN Research Institute, 1 Rue de la Noë, F-44321 Nantes FRANCE.*
E-mail: maryline.chetto@univ-nantes.fr

1.1 This research

In this paper, we focus on a system that consists of three components: a processing element with unique voltage and frequency, an energy harvester and a rechargeable energy storage unit.

We address the scheduling issue for uniprocessor platforms. We consider that the system allows for preemptions to process jobs with real-time constraints. Jobs must be executed in a timely manner and any deadline failure has to be anticipated in order to avoid an intolerable damage. Our processing model exhibits two major assumptions. Firstly, as in the most popular model of real-time computing, every job is characterized by a

release time, an execution requirement and a deadline. With such characterization, the job requires an amount of processor time equal to its execution requirement between its release time and its deadline. In our study, each job is characterized in addition by an energy requirement which is the amount of energy needed for its execution. We assume that the energy requirement of a job does not necessarily have to be proportional to its execution time. Secondly, as commonly assumed in the real-time energy harvesting literature [21], [25], the instantaneous consumption power of any job is no less than the incoming power from the harvesting unit i.e. the jobs are discharging.

Guaranteeing that all deadlines are met is one of the most important issues in Real-Time Energy Harvesting (RTEH) systems. We have to adopt a scheduling strategy that can always guarantee a predictable response time for every job even in the face of energy limitations. Another key consideration that affects power management and scheduling in RTEH systems is that instead of minimizing the energy consumption and maximizing the lifetime achieved as in classical battery operated devices, the system operates in an "energy neutral mode" by consuming only as much energy as harvested [16]. So the resulting problem we have to deal with is: How can we schedule the jobs so as to guarantee their timing constraints perpetually by suitably exploiting both the processor and the available ambient energy?

1.2 Prior research

Most prior research on scheduling of hard deadline jobs on a single processor computing system assumes that jobs have no energy requirements (see [5], [] for surveys). Dertouzos [11] shows that the Earliest Deadline First Algorithm (EDF) is optimal. EDF schedules at each instant of time t, that job ready for execution whose deadline is closest to t. But the problem with EDF is that it does not consider future jobs arrivals and their energy requirements. Energy shortage (i.e. the situation where the energy is not sufficient to execute jobs timely) should imperatively be anticipated to avoid deadline misses. In [7], we prove that EDF is no longer optimal for RTEH systems. Jobs are processed as soon as possible thus consuming the available energy greedily. We usualy say that EDF is work-conserving for released jobs. In other words, the scheduler never idles the processor while there is a job awaiting execution. Although non-competitive, EDF turns out to remain the best non-idling scheduler for uniprocessor RTEH platforms [7].

In an energy constrained system, it is sometimes necessary not to dispatch a ready job if it will prevent future jobs to meet their deadlines because of energy shortage. In [8], we show that no online scheduler can be optimal. With possible lookahead, we say that an online algorithm is lookahead-ld if ld is the length of time segment that the scheduler can foresee at any time [10]. Such a scheduler is also described as semi-online. We prove that optimality can be obtained only by semi-online schedulers with lookahead-D where D is the longest relative deadline of jobs in the application [8]. In other terms, an optimal scheduling algorithm that takes decisions at run time requires clairvoyance for at least D time units from any instant.

1.3 Contributions

The aim of this work is to provide an analysis in the context of dynamic-priority, preemptive, uniprocessor scheduling with energy harvesting considerations. Specifically, this paper integrates a general model for RTEH systems and extends some notions as processor demand, processor load and slack time to the energy domain with the notions of energy demand, energy load and slack energy. We report our findings concerning the study of the two following issues:

1) *Runtime scheduling*: Given a RTEH system that is known to be feasible, determine an online (or semi-online) scheduling algorithm that schedules the system to meet all deadlines.
2) *Feasibility test*: Given the specifications of a RTEH system, determine whether there exists a schedule that meets all deadlines. Performing a feasibility test provides a yes or no answer depending on whether the job set is feasible or not.

1.4 Outline

The remainder of the paper is organized as follows. The energy harvesting system model and assumptions are presented in Section 2. We give background materials in Section 3. New concepts and a new energy-aware scheduling algorithm, namely ED-H, are presented in Section 4. We prove the optimality of ED-H in Section 5 and we establish an exact feasibility test in Section 6. Section 7 focusses on practical considerations. Related works are described in Section 8. Section 9 summarizes this paper and provides directions for future works.

2 Model and Terminology

2.1 System Model

Hereafter, we describe the RTEH model that consists of a computing element, a set of jobs, an energy storage unit, an energy harvesting unit and an energy source (see Fig. 1).

2.1.1 Job model

We consider a set of real-time jobs that are executed on a single processing unit that supports only one operating frequency. Its energy consumption is only due to dynamic switching energy. Consequently, it consumes negligible energy in the idle state when it does not execute jobs. Jobs are processed exclusively with energy generated by the energy source. The set of jobs is denoted by $\tau = \{\tau_i, i = 1, \ldots, n\}$. All jobs can be preempted

and later resumed at any time with no time or energy loss associated with such preemption. Furthermore, jobs are independent of each other. A four-tuple (r_i, C_i, E_i, d_i) is associated with a job τ_i. In this characterization, job τ_i arrives at time r_i called release time, requires a worst case execution time of C_i time units and has a worst case energy consumption of E_i energy units. We assume that E_i is not necessarily proportional to C_i [13]. Executing any job of τ consumes at most e_{Max} units of energy during one unit time-slot and the actual energy consumption in the slot is not known beforehand. A deadline occurs at d_i units by which τ_i must have completed its execution. Let $d_{Max} = \max_{0 \leq i \leq n} d_i$ be the latest absolute deadline and $D = \max_{0 \leq i \leq n} (d_i - r_i)$ be the greatest relative deadline. The energy consumed by jobs on the time interval $[t_1, t_2)$ is denoted by $E_c(t_1, t_2)$. The energy consumed in any unit time-slot is no less than the energy produced in the same unit time-slot. We say that the jobs are "discharging" [3]. Consequently, the residual capacity of the energy storage unit is never increasing every time a job executes.

2.1.2 Energy production model

The energy produced by the source is not considered as controllable. It is characterized by an instantaneous charging rate $P_p(t)$ that incorporates all losses. The energy produced by such a power source in the time interval $[t_1, t_2)$ is given as $E_p(t_1, t_2) = \int_{t_1}^{t_2} P_p(t)dt$. We assume that the energy production times can overlap with the consumption times. While the source power is not necessarily a constant value, we assume that we can predict it accurately for near future with negligible time and energy cost.

2.1.3 Energy storage model

Our system uses an ideal energy storage unit (super-capacitor or battery) with a nominal capacity C. C is expressed in units of energy. The capacity may be less than the energy consumption of some jobs. Let us define $E(t)$ as the residual capacity at time t i.e. the energy level in the storage unit at t. Energy is wasted if the storage is fully charged at time t and we continue to charge it. For simplicity, $E(t) \approx C$ stands for $C \leq E(t) < C + e_{Max}$. In contrast, the energy storage is considered as fully discharged at time t if $0 \leq E(t) < e_{Max}$ denoted by $E(t) \approx 0$. The storage unit is fully charged initially (i.e. $E(0) = C$). The stored energy may be used at any time later and does not leak any energy over time.

2.2 Types of starvation

According to the RTEH model, a job τ_i can miss its deadline if one of the two following situations occurs:

- *time starvation*: when the job reaches its deadline at time t, its execution is incomplete because the time required to process it before deadline is not sufficient. There is available energy in the storage unit when the deadline violation occurs (i.e. $E(t) > 0$).

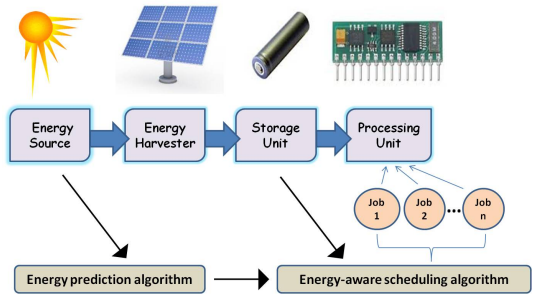

Fig. 1. A Real-Time Energy Harvesting System

- *energy starvation*: when the job reaches its deadline at time t, its execution is incomplete because the energy required to process it before deadline is not available. The energy in the storage unit is exhausted when the deadline violation occurs (i.e. $E(t) \approx 0$).

2.3 Terminology

We now give definitions we will be needing throughout the remainder of this paper.

Definition 1: A schedule Γ for τ is said to be *valid* if the deadlines of all jobs of τ are met in Γ, starting with a storage fully charged, with the energy generated by a given energy source.

Definition 2: A system is *feasible* if there exists at least one valid schedule for τ with the given energy source and energy storage unit. Otherwise, it is *infeasible*.

In infeasible RTEH systems, the limiting factors are either, both time and energy, only time or only energy. We focus here on feasible systems only.

As in the classical scheduling theory, we say that a scheduling algorithm is :

- *optimal* if it finds a valid schedule whenever one exists.
- *online* if it makes its decisions at run-time.
- *semi-online* if it is online with necessary lookahead on a certain time interval.
- *lookahead-ld* if it is semi-online with lookahead on ld time units.
- *idling* if it is allowed to keep the processor idle even when there are pending jobs. Otherwise, it is *non-idling or work-conserving*.
- *clairvoyant* if it has knowledge of the future.

We introduce a novel terminology which is peculiar to energy constrained computing systems.

Definition 3: A schedule Γ for τ is said to be *time-valid* if the deadlines of all jobs in τ are met in Γ, considering that $\forall i \in \{1, \ldots, n\}, E_i = 0$.

Definition 4: A job set τ is said to be *time-feasible* if there exists a time-valid schedule for τ.

Definition 5: A schedule Γ for τ is said to be *energy-valid* if the deadlines of all jobs in τ are met in Γ, considering that $\forall i \in \{1, \ldots, n\}, C_i = 0$.

Definition 6: A job set τ is said to be *energy-feasible* if there exists an energy-valid schedule for τ.

Definition 7: A scheduling algorithm A is said to be *energy-clairvoyant* if it needs knowledge of the future energy production to take its runtime decisions.

3 Background materials

3.1 EDF scheduling

EDF is probably the most famous dynamic priority scheduler [11], [17]. As a consequence of its optimality for preemptive uniprocessor scheduling of independent jobs, the run-time scheduling problem is perfectly solved if we assume there exists no additional constraints on the jobs. EDF is the scheduler of choice since any feasible set of jobs is guaranteed to have a valid EDF schedule.

In the most common way of EDF implementation, jobs are executed as soon as possible. In that version of EDF, the processor is never let inactive if at least one job is awaiting for execution. Such implementation has been called Earliest Deadline as Soon as possible (EDS) [6]. EDS is clearly an on-line scheduler since it solely needs timing parameters of the jobs which are currently ready for execution to take its dispatching decisions. By opposition, the jobs can be scheduled as late as possible according to the so-called Earliest Deadline as Late as possible (EDL) approach. Determination of the start time of the next job to execute requires knowledge of jobs that are currently ready and jobs that will arrive in the future as well. Such a version of EDF turns out to be semi-online since the EDL schedule build at any current time t_c needs information about jobs released between time t_c and time $t_c + D$ where D stands for the greatest relative deadline of the application. This makes of EDL an online lookahead-D scheduler [9]. Although the usual scheduling scheme is EDS, EDL is very often considered for processor idle time analysis.

3.2 Classical concepts

In this subsection, we recall definitions for real-time scheduling concepts.

3.2.1 Static analysis

Definition 8: The processor demand of a job set τ on the time interval $[t_1, t_2)$ is

$$h(t_1, t_2) = \sum_{t_1 \leq r_k, d_k \leq t_2} C_k \tag{1}$$

Definition 9: The static slack time of a job set τ on the time interval $[t_1, t_2)$ is

$$SST_\tau(t_1, t_2) = t_2 - t_1 - h(t_1, t_2) \tag{2}$$

$SST_\tau(t_1, t_2)$ gives the longest time that could be made available within $[t_1, t_2)$ after executing jobs of τ with release time at or after t_1 and deadline at or before t_2.

Definition 10: The static slack time of a job set τ, SST_τ, is

$$SST_\tau = \min_{0 \leq t_1 < t_2 \leq d_{Max}} SST_\tau(t_1, t_2) \tag{3}$$

Namely, the processor demand on $[t_1, t_2)$ gives the amount of execution time requested by all jobs with release time at or after t_1 and deadline before or at t_2. When the set of jobs utilizes the processor with a ratio less than 100%, there is unused processor time, hence the notion of slack time. The schedulability analysis for EDF needs to calculate the processor demand for every time interval starting with a release time and finishing with a deadline in order to check if there is an overflow in the interval. This amounts to computing the so-called static slack time $SST_\tau(t_1, t_2)$ with (3). In systems where jobs may arrive at unpredictable times, we have to perform the test (often called admission test) online so as to decide whether the new occurring job is authorized to enter into the system [5].

The processor load (as defined in the classical scheduling theory with no energy consideration) shows the maximum fraction of processor time requested in a given time interval.

Definition 11: The static processor load of a job set τ on the time interval $[t_1, t_2)$ is

$$USP_\tau(t_1, t_2) = \frac{h(t_1, t_2)}{t_2 - t_1} \tag{4}$$

$USP_\tau(t_1, t_2)$ gives the ratio of the total execution time to the length $t_2 - t_1$, considering all jobs which are released at or after t_1 with deadline at or before t_2.

Definition 12: The static processor load of a job set τ is

$$USP_\tau = \sup_{0 \leq t_1 < t_2 \leq d_{Max}} USP_\tau(t_1, t_2) \tag{5}$$

Consequently, if the static processor load of τ is greater than 1, there should not exist a feasible scheduling algorithm for τ. And we generally say that the system is overloaded [4]. More precisely, we will say that the system is in processor-overload.

3.2.2 Dynamic analysis

Let t_c be the current time in the schedule produced for the job set τ by a certain scheduling algorithm.

Definition 13: The slack time of a job τ_i at current time t_c is

$$ST_{\tau_i}(t_c) = d_i - t_c - h(t_c, d_i) - AT_i \tag{6}$$

where AT_i is the total remaining execution time of uncompleted jobs currently ready at t_c with deadline at or before d_i.

$ST_{\tau_i}(t_c)$ gives the available processor time after executing uncompleted jobs with deadlines at or before d_i.

Definition 14: The slack time of a job set τ at current time t_c is

$$ST_\tau(t_c) = \min_{d_i > t_c} ST_{\tau_i}(t_c) \quad (7)$$

We show in [6] that the slack time as computed with (7) represents the maximum continuous processor time that could be available from time t_c while still guaranteeing the deadlines of all the jobs. And it is obtained from the EDL schedule produced at current time t_c.

4 THE ED-H SCHEDULING ALGORITHM

4.1 Overview of the scheduling scheme

The intuition behind the scheduling algorithm we propose for the RTEH model is to run jobs according to the earliest deadline first rule. However, before authorizing a job to execute, the residual energy capacity of the storage unit must be sufficient to supply the awaiting highest priority job for at least the next unit time-slot. Furthermore, the energy consumption in that time-slot must guarantee the energy-feasibility of all future occurring jobs. This can be verified by considering their timing and energy requirements as well as the replenishment rate of the storage unit. If one of these conditions is not fulfilled, the processor has to idle so that the storage unit recharges sufficiently. Roughly speaking, this extension of EDF prevents energy starvation. Following the idea described above, we present a modified EDF that is dedicated to energy harvesting constrained jobs, called ED-H scheduling algorithm.

4.2 Concepts for the RTEH model

4.2.1 Static analysis

To formally present ED-H, we need to introduce novel concepts particularly helpful when analyzing the feasibility of a job set with both energy and deadline constraints: the energy demand, the slack energy and the energy load.

Let r_k, d_k and E_k be release time, deadline and worst case energy consumption of job τ_k respectively.

Definition 15: The energy demand of a job set τ on the time interval $[t_1, t_2)$ is

$$g(t_1, t_2) = \sum_{t_1 \le r_k, d_k \le t_2} E_k \quad (8)$$

Let $E_p(t_1, t_2)$ be the amount of energy that will be produced by the source between t_1 and t_2.

Definition 16: The static slack energy of a job set τ on the time interval $[t_1, t_2)$ is

$$SSE_\tau(t_1, t_2) = C + E_p(t_1, t_2) - g(t_1, t_2) \quad (9)$$

$SSE_\tau(t_1, t_2)$ gives the largest energy that could be made available within $[t_1, t_2)$ after executing jobs of τ with release time at or after t_1 and deadline at or before t_2.

Definition 17: The static slack energy of a job set τ is

$$SSE_\tau = \min_{0 \le t_1 < t_2 \le d_{Max}} SSE_\tau(t_1, t_2) \quad (10)$$

Intuitively, the static slack time of τ represents the length of the interval starting at any instant during which the processor could be idle continuously while still satisfying all the timing constraints of τ. It represents also the maximum processing surplus that could be accepted by τ at any instant. The static slack energy of τ represents the additional energy that could be consumed from any instant while still satisfying all the energy and timing constraints of τ. We now extend the concept of processor load to the energy domain.

Definition 18: The static energy load of a job set τ on the time interval $[t_1, t_2)$ is

$$USE_\tau(t_1, t_2) = \frac{g(t_1, t_2)}{C + E_p(t_1, t_2)} \quad (11)$$

Definition 19: The static energy load of a job set τ is

$$USE_\tau = \sup_{0 \le t_1 < t_2 \le d_{Max}} USE_\tau(t_1, t_2) \quad (12)$$

If the static energy load of τ is greater than 1, there should not exist a feasible scheduling algorithm for τ because of inevitable energy shortage in some time interval. We will say that the system is in energy-overload.

4.2.2 Dynamic analysis

Hereafter, for short, the slack time (respectively the slack energy) will actually refer to the dynamic slack time (respectively the dynamic slack energy) as regards current time t_c in the schedule produced for τ by a certain scheduling algorithm.

Definition 20: The slack energy of a job τ_i at current time t_c is

$$SE_{\tau_i}(t_c) = E(t_c) + E_p(t_c, d_i) - g(t_c, d_i) \quad (13)$$

Clearly, $SE_{\tau_i}(t_c)$ represents the maximum energy that could be consumed within $[t_c, d_i)$ while guaranteeing enough energy for jobs released at or after t_c and deadline at or before d_i. In other words, if there exists some job τ_i such that $SE_{\tau_i}(t_c) = 0$, executing any job with deadline after d_i between t_c and d_i will provoque energy starvation for τ_i.

Conventional EDF is greedy since it executes jobs as soon as possible and spends the stored energy disregarding needs of future jobs. So, a scheduler that withdraws energy from the storage unit should not cause future energy starvation. If we assume jobs to be scheduled according to the earliest deadline rule, energy starvation on a job say τ_i can only be caused by a job, say τ_j which executes before the release of τ_i

such that $d_j > d_i$. Let us note that energy starvation of τ_i caused by τ_j with $d_j \leq d_i$ could not be avoided. Intuitively, clairvoyance on jobs arrivals and energy production will help EDF to anticipate possible energy starvation and deadline violation. The main principle of ED-H is to authorize job executions as long as no future starvation could occur. This leads us to define the so-called *preemption slack energy* for current time t_c as the maximum energy that could be consumed by the currently active job while still guaranteeing energy feasibility for jobs that may preempt it.

Definition 21: Let d be the deadline of the active job at current time t_c. The preemption slack energy of a job set τ at t_c is

$$PSE_\tau(t_c) = \min_{t_c < r_i < d_i < d} SE_{\tau_i}(t_c) \quad (14)$$

4.3 Description of the ED-H scheduler

In what follows, we consider a given set of jobs that is known to be feasible for the RTEH model. Let $L_r(t_c)$ be the list of uncompleted jobs ready for execution at t_c. The ED-H scheduling algorithm obeys the following rules:

- **Rule 1:** The EDF priority order is used to select the future running job in $L_r(t_c)$.
- **Rule 2:** The processor is imperatively idle in $[t_c, t_c + 1)$ if $L_r(t_c) = \emptyset$.
- **Rule 3:** The processor is imperatively idle in $[t_c, t_c + 1)$ if $L_r(t_c) \neq \emptyset$ and one of the following conditions is satisfied:
  1) $E(t_c) \approx 0$.
  2) $PSE_\tau(t_c) \approx 0$
- **Rule 4:** The processor is imperatively busy in $[t_c, t_c + 1)$ if $L_r(t_c) \neq \emptyset$ and one of the following conditions is satisfied:
  1) $E(t_c) \approx C$.
  2) $ST_\tau(t_c) = 0$
- **Rule 5:** The processor can equally be idle or busy in $[t_c, t_c + 1)$ if $L_r(t_c) \neq \emptyset$, $0 < E(t_c) < C$, $ST_\tau(t_c) > 0$ and $PSE_\tau(t_c) > 0$.

Rules 3.1 and 3.2 say that the processor cannot be active if either the energy storage unit is deplenished or executing any job would prevent at least one future job from beeing executed timely because of energy starvation i.e. the system has no preemption slack energy at t_c. Rules 4.1 and 4.2 say that the processor cannot be inactive if either the energy storage unit is fully replenished or making the processor idle would prevent at least one job from beeing executed timely because of time starvation i.e. the system has no slack time at t_c. When the storage unit is neither full nor empty and the system has both slack time and preemption slack energy, rule 5 says that the scheduler may decide on the processor state.
We notice that we never dispatch jobs when there is no energy. We start charging the storage unit when, either it is empty or there is not enough energy to guarantee the feasible execution of all future occurring jobs. The charging process is flexible since it authorizes to charge the storage unit during any time period provided there is slack time and the storage unit has not replenished. We only waste recharging power when there are no ready jobs and the storage unit is full.

The above description of ED-H forgets the case where the energy storage unit is fully replenished at t_c (i.e. $C \leq E(t_c) < C + e_{Max}$) and the system has no preemption slack energy (i.e. $0 \leq PSE_\tau(t_c) < e_{Max}$). In order not to waste energy by idling the processor, we may advance the jobs and execute the highest priority job in $[t_c, t_c + 1)$. Thus, at most e_{Max} energy units are consumed after which the processor stays idle so that the storage unit be fully replenished again. Consequently, the ED-H scheduler continuously switches from the idle state to the active state so that the effective average consumption energy be equal to the production energy with accuracy within e_{Max}. This results in wasted energy less than e_{Max} units until the completion of the job having a zero slack energy. In contrast to the assumption that is claimed in [25], the instantaneous consumption power of a running job cannot be adjusted to fit the incoming environmental power.

We may derive various implementations from the above ED-H scheme. It depends on the application of rule 5. Jobs can be processed ASAP, ALAP or mixture of ASAP and ALAP strategies. The rule for deciding when to start and stop recharging with inserted idle time periods determines the resulting ED-H variant. The only condition is to prevent from negative slack time, negative slack energy and energy wasting at every time instant.

Example 1: Let us consider two jobs τ_1 and τ_2 with release times $r_1 = 0$, $r_2 = 1$, execution times $C_1 = 1$, $C_2 = 3$, energy consumptions $E_1 = 2$, $E_2 = 8$, absolute deadlines $d_1 = 8$, $d_2 = 6$. The energy storage unit has a capacity $C = 6$ and we assume that $E(0) = 4$. The energy production power is constant with $P_p = 1$. The maximum instantaneous consumption power is known by $e_{Max} = 3$.
First, by applying equation (13) we compute $SE_{\tau_2}(0) = E(0) + E_p(0, d_2) - g(0, d_2) = 4 + 6 - 8 = 2$ as τ_2 is the only job with release time after time 0 and deadline before d_1. Clearly, $SE_{\tau_2}(0)$ represents the maximum energy that can be consumed by jobs from time 0 until the start time of τ_2 without injuring the energy feasibility of τ_2. Equation (14) enables us to obtain $PSE_\tau(0) = 2$ i.e. $PSE_\tau(0) \approx 0$ since $e_{Max} = 3$. From rule 3.2, the processor idles imperatively. Let us choose to let the processor idle as long as possible. We compute the slack times of τ_1 and τ_2 by formula (6). $ST_{\tau_1}(0) = d_1 - h(0, d_1) - AT_1 = 8 - (1 + 3) - 0 = 4$ and $ST_{\tau_2}(0) = d_2 - h(0, d_2) - AT_2 = 6 - 3 - 0 = 3$. Hence, formula (7) gives $ST_\tau(0) = 3$. Let us compute the time instant, say t_f when the storage will be fully replenished.

t_f satisfies the following equation: $E(0) + E_p(0, t_f) = C$ which leads to $t_f = 2$. The processor is let idle until time 2 where $E(2) = 6$. The processor starts execute the highest priority job τ_2. We may decide to execute it as soon as possible until completion. τ_2 completes at time 5 where $E(5) = 1$. As $E(5) \approx 0$, the processor imperatively idles. We may decide to recharge until there is no more slack time, say at time instant $d_1 - C_1$ i.e. 7 where $E(7) = 3$. τ_1 completes at deadline where $E(8) = 1$ (see Fig. 2). Notice that EDF would execute τ_1 first and lead to energy starvation for τ_2 at time 2.

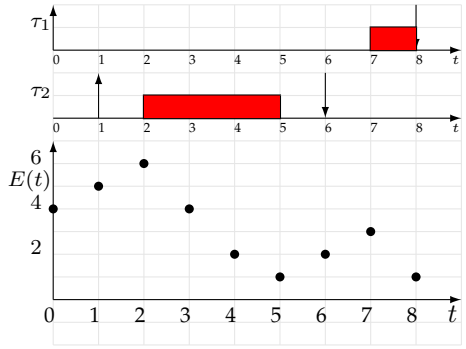

Fig. 2. ED-H scheduling

5 Properties of ED-H scheduling

5.1 Optimality analysis

We state the optimality of ED-H by proving that if ED-H cannot schedule a given job set τ, then no other scheduling algorithm is able to schedule it. We assume that the deadline at d_1 of job τ_1 is missed and d_1 is the first deadline of τ that is missed in the ED-H schedule. This violation is due to one of the two following reasons: either job τ_1 lacks time (Lemma 1) or job τ_1 lacks energy (Lemma 2) to complete its execution before or at deadline d_1. The time starvation case is when deadline d_1 is missed with the storage not exhausted at d_1. The energy starvation case is when the storage is exhausted at d_1 and τ_1 is not completed.

Lemma 1: If d_1 is missed in the ED-H schedule because of time starvation, there exists a time instant t such that $h(t, d_1) > d_1 - t$ and no schedule exists where d_1 and all earlier deadlines are met.

Proof: Recall that we have to consider the time starvation case where d_1 is missed with $E(d_1) > 0$. Let t_0 be the latest time before d_1 where the processor is idle. Consequently, the processor is continuously busy between t_0 and d_1. We have to examine the two following cases.

Case 1: There is no ready job at time t_0.

Consequently, t_0 coincides with the arrival of a job, say τ_2 with release time $r_2 = t_0$ that verifies $r_2 \leq r_1$.

Case 1a: $d_2 \leq d_1$. τ_2 is entirely processed before d_2 because d_1 is the first deadline to be violated and jobs are scheduled according to the earliest deadline rule in ED-H. We may have the following two cases.

Case 1a1: According to rule 4.2, the processor is busy at time r_2 because there is no slack time at r_2 i.e. $ST_\tau(r_2) = 0$. Thus, the slack time at r_2 of τ_1 is no less than 0. Consequently, the processor demand on the time interval $[r_2, d_1)$, given by $h(r_2, d_1) = \sum_{r_2 \leq r_k, d_k \leq d_1} C_k$ is no less than $d_1 - r_2$. This contradicts that d_1 is violated. And no other scheduling algorithm can produce a valid schedule on $[r_2, d_1)$.

Case 1a2: According to rule 5, the processor is busy at time r_2 because $0 < E(r_2) < C$, $ST_\tau(r_2) > 0$ and $PSE_\tau(r_2) > 0$. Consequently, $ST_{\tau_1}(r_2) > 0$ i.e. $h(r_2, d_1) = \sum_{r_2 \leq r_k, d_k \leq d_1} C_k$ is no more than $d_1 - r_2$. This contradicts that d_1 is violated.

Case 1b: $d_2 > d_1$. No job released before r_1 with deadline greater than d_1 including job τ_2 is executed within $[r_1, d_1)$ because jobs are scheduled according to the earliest deadline rule. Consequently, the maximum computation time required by jobs in $[r_1, d_1)$ is equal to the processor demand $h(r_1, d_1) = \sum_{r_1 \leq r_k, d_k \leq d_1} C_k$. Since d_1 is violated, necessarily $h(r_1, d_1) > d_1 - r_1$ because jobs are ordered by the earliest deadline rule. And no other scheduling algorithm can produce a valid schedule.

Case 2: There is at least one ready job at time t_0.

The processor stops to be idle at time t_0 if

Case 2a: The energy storage is fully replenished i.e. $E(t_0) = C$ from rule 4.1.

Case 2b: The slack time of τ becomes zero i.e. $ST_\tau(t_0) = 0$ from rule 4.2.

Case 2c: We may stop the processor at t_0 by rule 5 if $0 < E(t_0) < C$, $ST_\tau(t_0) > 0$ and $PSE_\tau(t_0) > 0$.

Whatever the stop condition, $ST_\tau(t_0) \geq 0$. Since $ST_{\tau_1}(t_0) \geq ST_\tau(t_0)$, $d_1 - t_0 \geq h(t_0, d_1)$. $h(t_0, d_1)$ represents the total amount of computation time required by jobs with deadline at or before d_1 which are ready at time t_0 and released within $[t_0, d_1)$. Since jobs are scheduled according to the earliest deadline first rule in ED-H, and there is no idle time in $[t_0, d_1)$, $d_1 - t_0 > h(t_0, d_1)$ contradicts that d_1 is missed. Let us note that the special case where the storage is fully replenished and there is no slack energy is treated as rule 4.1. □

Lemma 1 states that there exists some interval $[t, d_1)$ where the processor demand $h(t, d_1)$ is higher than the maximum available processor time equal to $d_1 - t$ that could be available in that interval.

Lemma 2: If d_1 is missed in the ED-H schedule because of energy starvation there exists a time instant t such that $g(t, d_1) > C + E_p(t, d_1)$ and no schedule exists where d_1 and all earlier deadlines are met.

Proof: Recall that we have to consider the energy starvation case where d_1 is missed with $E(d_1) = 0$. Let t_0 be the latest time before d_1 where a job with deadline after d_1 releases, no other job is ready just before t_0 and the energy storage unit is fully charged i.e. $E(t_0) = C$.

The initialization time can be such time. The processor is idle within $[t_0 - 1, t_0)$ since no jobs are ready. As no energy is wasted except when there are no ready jobs, the processor is busy at least from time t_0 to $t_0 + 1$. We consider two cases:
Case 1: No job with deadline after d_1 executes within $[t_0, d_1)$. Consequently, all the jobs that execute within $[t_0, d_1)$ have release time at or after t_0 and deadline at or before d_1. The amount of energy required by these jobs is $g(t_0, d_1)$. As τ is feasible, $g(t_0, d_1)$ is no more than the maximum storable energy plus all the incoming energy i.e. $C + E_p(t_0, d_1)$. As $E(t_0) = C$, we conclude that all jobs ready within $[t_0, d_1)$ can be executed with no energy starvation which contradicts the deadline violation at d_1 with $E(d_1) = 0$.
Case 2: At least one job with deadline after d_1 executes within $[t_0, d_1)$.
Let t_2 be the latest time where a job, say τ_2, with deadline after d_1 is executed. As d_1 is lower than d_2 and jobs are executed according to the earliest deadline rule in ED-H, we have $r_2 < r_1$. At time t_2, one of the following situations occurs.

Case 2a: The processor is busy all the times in $[t_2, d_1)$. τ_2 is preempted by a higher priority job, say τ_3, with $d_3 \leq d_1$. From rule 4.2, $PSE_\tau(r_3) > 0$ which implies that $SE_{\tau_1}(r_3) > 0$ and in consequence $g(r_3, d_1) < E(r_3) + E_p(r_3, d_1)$. All jobs that are executed within $[r_3, d_1)$ have release time at or after r_3 and deadline at or before d_1. Consequently, the amount of energy they require is at most $g(r_3, d_1)$. That contradicts deadline violation and $E(d_1) = 0$.

Case 2b: The processor is idle in $[t_3 - 1, t_3)$ with $t_3 > t_2$ and busy all the times in $[t_3, d_1)$.
The processor stops idle at time t_3 imperatively by rule 4.1 if $E(t_3) = C$. By hypothesis, there is no job waiting with deadline at or before d_1 at t_3 because t_0 is the latest one. Furthermore, no job with deadline after d_1 is executed after t_2 and consequently after t_3. In order not to waste energy, all the energy which arrives from the source is used to advance jobs with deadline after d_1. The processor continuously commutes from active state to inactive state. The storage is maintained at maximum level until τ_1 releases. Consequently, we have $E(r_1) = C$. As τ is feasible, $g(r_1, d_1) \leq C + E_p(r_1, d_1)$. Thus, $E(r_1) + E_p(r_1, d_1) \geq g(r_1, d_1)$. That contradicts deadline violation and $E(d_1) = 0$. □

Lemma 2 states that there exists some interval $[t, d_1)$ where the energy demand $g(t, d_1)$ is higher than the maximum energy equal to $C + E_p(t, d_1)$ that could be available in $[t, d_1)$. We may draw Theorem 1, a major result for uniprocessor scheduling with real time and energy harvesting constraints.

Theorem 1: The ED-H scheduling algorithm is optimal for the RTEH model.
Proof: According to Lemma 1, if ED-H cannot schedule a given set of jobs τ because of time starvation, then no other scheduling algorithm is able to schedule it. According to Lemma 2, if ED-H cannot schedule a given set of jobs τ because of energy starvation, then no other scheduling algorithm is able to schedule it. As a conclusion, if ED-H cannot schedule a given set of jobs τ for time or/and energy starvation, then no other scheduling algorithm is able to schedule it because time starvation and energy starvation are the only two reasons for deadline violations. And we conclude that ED-H is optimal. □

Optimality signifies that ED-H can produce a valid schedule as long as there is no time interval with a length lower than the processor demand and no time interval where the energy demand is greater than the available energy. In other words, any job set which is neither processor-overloaded nor energy-overloaded should be feasible i.e. schedulable by ED-H.

5.2 Clairvoyance analysis

Now, let us show that ED-H is semi-online, more exactly online with lookahead-D. In other words, taking a decision at current time t_c requires knowledge of the energy incoming and jobs'arrivals on the next D unit time-slots at the most. Recall that from our prior results [8], we know that no online scheduling algorithm can be optimal if the lookahead parameter is less than D.

Lemma 3: Computation of the slack time at runtime can be achieved by an online lookahead-D algorithm.
Proof: From formulae given in [9], $ST_\tau(t_c)$ can be calculated from timing parameters of jobs with release time lower than the deadline of the uncompleted highest priority job ready at t_c. An upper bound on the relative deadline of this job is D. □

Lemma 4: Computation of the slack energy at runtime can be achieved by an online lookahead-D algorithm.
Proof: The slack energy of a job τ_i given by (9) represents the maximum available energy in $[t_c, d_i)$ after execution of higher priority jobs i.e. jobs with release time after t_c and deadline at or before d_i. Consequently, the slack energy of τ_i is the largest energy that could be consumed in $[t_c, d_i)$ by lower priority jobs i.e. jobs with a deadline greater than d_i. Let d be the deadline of the highest priority job ready at time t_c. Clearly, $d - t_c < D$. In order to calculate the amount of energy that could be consumed by this job from time t_c (i.e. the slack energy of the system at t_c), we only compute the slack energy of the jobs that may preempt it i.e. the ones with release time after t_c and deadline less than d. And all these jobs have a relative deadline at most equal to D. □

Theorem 2: The ED-H scheduling algorithm is online lookahead-D.
Proof: We prove that every runtime dispatching decision requires clairvoyance for the next D unit time-slots at

most. According to ED-H, clairvoyance is required at time t_c for computing $ST_\tau(t_c)$ or $PSE_\tau(t_c)$ only. From Lemma 3 and Lemma 4, ED-H is clearly an online lookahead-D scheduler. □

One challenge is to make possible an accurate assessment of the ambient energy at any current time for at least the next D unit time-slots. In some applications, the energy source may be modeled and the actual energy can be calculated off-line. In some other applications, we can determine online a lower bound on the energy produced on sliding windows. These approximations come from appropriate measurements and prediction methods [22].

6 Feasibility test

This section is concerned with the algorithm which, given a job set τ is capable of answering the question: Is τ feasible? Notice that τ is feasible if and only if there exists at least one schedule for which all the deadlines can be met, given the capacity C of the energy storage, and the source power $P_p(t)$, for $0 \leq t \leq d_{Max}$.

As we proved that ED-H is optimal, we have to obtain conditions upon the RTEH system under which the ED-H scheduling algorithm guarantees to meet all deadlines for this set of jobs. We will show that the schedulability test for ED-H reduces to test time-feasibility and energy-feasibility separately.

6.1 Time-feasibility

We first consider the time constraint case. In this case all jobs have processing requirements only, i.e. for every job $\tau_i \in \tau$, $E_i = 0$. We recall a basic result in the classical scheduling theory which relates the processor demand in each time interval with the length of this interval. Lemma 5 below specifies a neccesary and sufficient time-feasibility condition which comes down to a necessary feasibility condition in the RTEH model.

Lemma 5: τ is time-feasible if and only if

$$SST_\tau \geq 0 \tag{15}$$

Proof: "If": Directly follows from Lemma 1.
"Only If": See [31] □

Let us note that it suffices to compute the static slack time for all intervals starting at a release time and finishing at a deadline. Thus, the complexity of the time-feasibility test is $O(n^2)$ since n^2 time intervals must be tested. While Lemma 5 is based on the slack time approach, the following time-feasibility test provides the same condition with the load approach:

Lemma 6: τ is time-feasible if and only if

$$USP_\tau \leq 1 \tag{16}$$

Informally, inequality (15) or (16) can be though of as a requirement that the system not be processor-overloaded in any time interval.

6.2 Energy-feasibility

Now, we consider the energy constraint case where jobs have energy requirements only, i.e. for every job $\tau_i \in \tau$, $C_i = 0$. In other words, each job executes instantaneously.

Lemma 7: τ is energy-feasible if and only if

$$SSE_\tau \geq 0 \tag{17}$$

Proof: "If": Directly follows from Lemma 2.
"Only If": Since τ is energy-feasible, let us consider an energy-valid schedule produced within $[0, d_{Max})$. The amount of energy demanded in each interval of time $[t_1, t_2)$, $g(t_1, t_2)$, is necessarily less than or equal to the actual energy available in $[t_1, t_2)$ given by $E(t_1) + E_p(t_1, t_2)$. An upper bound on $E(t_1)$ is the maximum storable energy at time t_1, that is C. Consequently, $g(t_1, t_2)$ is lower than or equal to $C + E_p(t_1, t_2)$. This leads to $\forall [t_1, t_2) \subset [0, d_{Max})$, $g(t_1, t_2) \leq C + E_p(t_1, t_2)$ i.e. $SSE_\tau(t_1, t_2) \geq 0$. Thus, $SSE_\tau \geq 0$. □

The energy-feasibility test can be expressed with the energy load based formulation:

Lemma 8: τ is energy-feasible if and only if

$$UE_\tau \leq 1 \tag{18}$$

Proof: As proof of Lemma 8 since $SSE_\tau(t_1, t_2) \geq 0$ amounts to $UE_\tau(t_1, t_2) \leq 1$. □

We assume that the prediction of the ambient energy on every time interval provides a fixed number of values. Thus, the complexity of the energy-feasibility test is $O(n^2)$ since n^2 intervals must be tested.

6.3 ED-H schedulability test

Hereafter, we present a test for the purpose of validating that a given job set can indeed meet its deadlines, be given the capacity of the energy storage unit and the incoming power function. We give a necessary and sufficient condition for ED-H schedulability and feasibility by virtue of optimality.

Theorem 3: τ is feasible if and only if

$$SST_\tau \geq 0 \text{ and } SSE_\tau \geq 0 \tag{19}$$

Proof: "Only if": Suppose that τ is feasible. Thus, τ is time-feasible and energy feasible. From constraint (15) in Lemma 5 and constraint (17) in Lemma 7, it is the case that constraint (19) is satisfied.
"If": We suppose that constraint (19) is satisfied and τ is not schedulable by ED-H. Let us show a contradiction. First, we assume that τ is not schedulable by ED-H because of time starvation. Lemma 1 states that there exists a time interval $[t_0, d_1)$ such that $h(t_0, d_1) > d_1 - t_0$

i.e. $d_1 - t_0 - h(t_0, d_1) < 0$. Thus, $SST_\tau < 0$ and condition (19) in Theorem 3 is violated. Second, we assume that τ is not schedulable by ED-H because of energy starvation. Lemma 2 states that there exists a time interval $[t_0, d_1)$ such that $g(t_0, d_1) > C + E_p(t_0, d_1)$ i.e. $C + E_p(t_0, d_1) - g(t_0, d_1) < 0$. Thus, $SSE_\tau < 0$ and condition 19 in Theorem 3 is violated. □

In other terms, Theorem 3 states that τ is feasible if and only if τ is time-feasible and energy-feasible.

A criterion that can be used to measure the performance of scheduling algorithms is the schedulable utilization. Similarly to [18], we define the schedulable processor utilization (respectively energy utilization) of a scheduling algorithm as follows: A scheduling algorithm can produce a valid schedule of any job set if the total processor load (respectively energy load) is equal to or less than the schedulable processor utilization (respectively energy utilization) of the algorithm. Consequently, the higher the schedulable processor utilization and the higher the schedulable energy utilization, the better the scheduling scheme.

The following theorem gives the feasibility test with the load based formulation.

Theorem 4: τ is feasible if and only if

$$UP_\tau \leq 1 \text{ and } UE_\tau \leq 1 \tag{20}$$

Theorem 4 clearly shows that ED-H provides the highest possible schedulable processor utilization and energy utilization. In other words, the processor can be continuously busy in certain time interval(s), consuming both the entire capacity of the storage unit and all the energy drawn from the environment while meeting all the deadlines. This explains why no scheduling algorithm can be better than ED-H.

Remark: In [25], Moser et al present an optimal solution (called LSA) to the same scheduling problem as we do here. Nevertheless, their processing model assumes that the energy consumption and the execution time of every job behave proportional. Let constant P_{max} such that $C_i = \frac{E_i}{P_{max}}$. Consequently, $g(t_1, t_2) = P_{max} \times h(t_1, t_2)$. It can easily be verified that the above feasibility test given by equations (19) comes down for this special case to the LSA schedulability test reported in [25].

7 Practical Considerations

In this section, we consider the implications of the ED-H scheduling algorithm for practical harvesting system design. Most of prediction methods track past harvested energy and use them to predict future energy availability [15]. Various prediction models have been investigated in [21]. For example, the *moving average* technique predicts future values based on the averages of the past observations, giving equal weight to all past data. The *exponential smoothing* technique uses different weight factors for past values that depend on the distance from the current time. The experiment reported by Liu et al. highlight the important impact of the energy prediction technique on the resulting performance measured in terms of deadline miss rate. It is obvious that higher the accuracy of the prediction mechanism, higher the time and memory overhead incurred by its implementation. Nonetheless, more limited its negative impact on the performance of the actual ED-H scheduler compared with theoretical optimal ED-H.

RTEH systems encompass various application areas. Their common characteristics are the periodic activities that mostly involve sampling a sensor, processing the sensed value, transmitting data, sending data to an actuator, etc. It follows that the parameters of jobs are well known before the system becomes operational since they are the instances of a fixed set of periodic taks.

Feasibility checking aims to predict whether time and energy will be enough to meet the timing requirements. Generally, the basic steps in the design of real-time systems are to perform an off-line check and to schedule and dispatch the jobs at runtime. For RTEH systems, the checking can be done off-line when all the jobs are instances of periodic tasks and the energy profile is precisely characterized for all the application lifetime such as constant over time. Otherwise, the checking should be realized at runtime in dependance with the horizon of the prediction technique.

Under the planning-based approach, time is divided in windows. The schedulability checking is performed at runtime for each window. It detects future deadline miss and anticipates by deciding how to manage the transient shortage of energy or processor overload that will occur in the next window. For example, we may trade off quality for timeliness by executing back-up jobs with shorter execution times and lower energy requirements [6]. A best-effort technique with no online check would lead to deadline failures with both time and energy wasting.

8 Related Work

The technical challenges to achieve energy autonomy and to make energy harvesting systems work effectively are described in [15], [16]. Works of Kansal et al. focus on solar energy harvesting sensor nodes. A framework is presented for dynamically adapting the duty cycle of the node, using measurement of the deviations in energy values from an estimated model of the energy source. The method was evaluated in 2005 on the Heliomote platform. However, the above works do not target at real-time applications where jobs execute with deadline constraints.

The problem of scheduling RTEH systems has gained little attention. In 2001, Allavena et al. [3] propose an optimal claivoyant scheduler restricted to a set of independent periodic tasks in a frame (all the tasks have

the same deadline and repetition period). The power scavenged by the energy source is constant and all jobs consume energy at a constant rate. This work has been extended to processors provided with DVFS technology and tasks with multiple versions [30].

In 2006, Moser et al. give a variant of EDF, called Lazy Scheduling Algorithm (LSA) [25]. LSA provides an optimal online solution with time lookahead-D on the energy incoming. Theory and simulations show that LSA outperforms EDF up to 45% in terms of achievable capacity savings [26]. More recent research works can be found in [19], [20], [21], [23] where extensions of LSA with DVFS permit to improve the deadline miss rate and energy saving. The so-called HA-DVFS algorithm exploits the slack time of the jobs to slow down their execution whenever possible or speeds up their execution in order to use overflowing harvested energy. The proposed EDF based power management techniques use various slack management algorithms that permit to consume the slack time by inserting idle periods and thus refilling the energy storage. The above studies use a similar model of the power source as we do but consider that the total energy consumption of every job is proportional to its execution time (cf remark in Section 6.3). In real facts, instantaneous power consumed by jobs varies along time depending on circuitry and devices required by their execution (e.g. the power consumption when transmitting data differs from when receiving data) [13], [28]. This observation has motivated the study reported here which applies to a general model.

The issue of dynamic priority scheduling of periodic task sets was also investigated in [12]. The EDeg heuristic that uses slack time and slack energy concepts is presented with no schedulability test and no formal performance assessment for it. EDeg makes a significant performance enhancement in comparison to EDF in the light of simulation results even if the harvested energy is needlessly wasted in some specific situations.

Research on fixed priority scheduling for RTEH systems is recent [1], [2]. An optimal scheduler, namely PFPasap, is proposed. However, only periodic tasks as well as a constant source power are considered.

9 Conclusion and Future works

There are growing needs for energy harvesting capabilities in a variety of applications such as military, health and environmental (battlefield surveillance, telemonitoring of human physiological data, forest fire detection, etc.) due to the limitation of traditional battery power. Achieving perpetual and self-sustaining operation of an embedded device is becoming an important topic of research. The main issue is to perform, in one hand effective utilization of processor time and, in the other hand effective utilization of available and future energy resource with avoiding energy depletion.

This paper has considered an online scheduling problem arising from any kind of real-time energy harvesting system which must schedule jobs under deadline constraints. Most prior theoretical research concerning this problem imposed that either the jobs are instances of periodic tasks or there is a linear relationship between execution time and energy consumption. In this work, we have removed these assumptions.

We have presented a novel energy-aware scheduling algorithm, namely ED-H, proved to be optimal and appropriate for the scheduling of real time jobs in general. ED-H is idling and takes decisions that result from slack time and slack energy online computations. This makes ED-H very flexible in the determination of busy v.s inactive periods. We have formulated a schedulability test for ED-H which decouples the time and energy constraints. This test says us that ED-H is able to feasibly schedule any job set as long as both the processor load and the energy load are no more than one. We outline that ED-H could be applied to a set of jobs composed from the instances of periodic or/and sporadic tasks. Moreover, our approach to job scheduling does not require a model of energy replenishment and applies to any energy source provided the prediction be possible at runtime.

Our algorithm is also applicable to a wide range of practical problems which are not necessarily in the field of computer science. For example, it could be employed for dynamic power management on large machines such as electric vehicles where the harvested energy must be provided for motors notably.

For future work, we will explore:

- adaptation of the proposed scheduling scheme to fixed priority environments. This would suppose to modify computation formulae for the slack time and the slack energy accordingly.
- extension of ED-H to support DVFS technology.
- dimensionning i.e. calculation of the smallest capacity for the energy storage unit, the smallest harvester, etc. This kind of analysis refers to sensitivity analysis. Determining the so-called critical scaling factor [32] is obviously of economic interest in the design of any energy harvesting embedded device.

References

[1] Y. Abdeddaim, D. Masson. *Real-Time Scheduling of Energy Harvesting Embedded Systems with Timed Automata*, Proc. of 18th IEEE Int. Conference on Embedded and Real-Time Computing Systems and Applications, 2012.

[2] Y. Abdeddaim, Y. Chandarli, D.Masson. *The Optimality of PFPasap Algorithm for Fixed-Priority Energy-Harvesting Real-Time Systems*, Proc. of 25th Euromicro Conference on Real-Time Systems, 2013.

[3] A. Allavena, D. Mosse. *Scheduling of Frame-based Embedded Systems with Rechargeable Batteries*, Workshop on Power Management for Real-Time and Embedded Systems, 2001.

[4] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. *On the competitiveness of on-line real-time task scheduling*, Proc. of the Real-Time Systems Symposium, pp 106-115, 1991.

[5] G. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications.*, Springer, Berlin, 2005.

[6] H. Chetto, M. Chetto. *Some Results of the Earliest Deadline Scheduling Algorithm*, IEEE Transactions on Software Engineering, Volume 15, Issue 10, pp. 1261-1270, 1989.

[7] M. Chetto, A. Queudet. *A Note on EDF Scheduling for Real-Time Energy Harvesting Systems*, IEEE Transactions on Computers, DOI.ieeecomputersociety.org/10.1109/TC.2013.21, january 2013.
[8] M. Chetto, A. Queudet. *Clairvoyance and Online Scheduling in Real-Time Energy Harvesting Systems*, Real-Time Systems, DOI: 10.1007/s11241-013-9193-1, 2013.
[9] M.Chetto-Silly. *The EDL Server for Scheduling Periodic and Soft Aperiodic Tasks with Resource Constraints*, Real-Time Systems 17(1), pp 87-111, 1999.
[10] B. Coleman, W. Mao. *Lookahead scheduling in a real-time context*, Proc. of the 6th Int. Conference on Computer Science and Informatics, pp. 205-209, 2002.
[11] M.-L. Dertouzos. *Control Robotics: The Procedural Control of Physical Processes*, Proc. of Int. Federation for Information Processing Congress, 1974.
[12] H. El Ghor, M. Chetto, R. Hage Chehade. *A real-time scheduling framework for embedded systems with environmental energy harvesting*, Journal of Computers and Electrical Engineering, Volume 37 Issue 4, pp. 498-510, 2011.
[13] R. Jayaseelan, T. Mitra, X. Li. *Estimating the Worst-Case Energy Consumption of Embedded Software*, Proc. of 12th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 81-90, 2006.
[14] X. Jiang, J. Polastre, D.-E. Culler. *Perpetuel Environmentally Powered Sensor Networks*, Proc. of the 4th int. symposium on Information processing in sensor networks, pp. 463-468, 2005.
[15] A. Kansal, J. Hsu, S. Zahedi, M.B. Srivastava. *Power Management in Energy Harvesting Sensor Networks*, ACM Transactions on Embedded Computing Systems, Vol. 6, No. 4, 2007.
[16] A. Kansal, J. Hsu. *Harvesting aware Power Management for Sensor Networks*, Proc. of ACM/IEEE Design Automation Conference, pp. 651-656, 2006.
[17] C.-L. Liu, J.-W. Layland. *Scheduling algorithms for multiprogramming in a hard real-time environment*. Journal of the Association for Computing Machinery, Volume 20, Issue 1, pp. 46-61, 1973.
[18] J. W. S. Liu. *Real-Time Systems*, Prentice Hall, 592 pages, 2000.
[19] S. Liu, Q. Qiu, Q. Wu. *Energy Aware Dynamic Voltage and Frequency Selection for Real-Time Systems with Energy Harvesting*, Proc. of Design, Automation and Test in Europe, pp. 236-241, 2008.
[20] S. Liu, Q. Wu, Q. Qiu. *An adaptive scheduling and voltage/frequency selection algorithm for real-time energy harvesting systems*, Proc. of ACM/IEEE Design Automation Conference, pp. 782-787, 2009.
[21] S. Liu, J. Lu, Q. Wu, Q. Qiu. *Harvesting-Aware Power Management for Real-Time Systems with Renewable Energy*, IEEE Transactions on Very Large Scale Integration Systems, pp. 1-14, 2011.
[22] J. Lu, S. Liu, Q. Wu, Q. Qiu. *Accurate modeling and prediction of energy availability in energy harvesting real-time embedded systems*, Proc. of Int. Conference on Green Computing, pp. 469-476, 2010.
[23] J. Lu, Q.Qiu. *Scheduling and mapping of periodic tasks on multi-core embedded systems with energy harvesting*, Proc. of Int. Conference on Green Computing, pp. 1-6, 2011.
[24] Y.-H. Lu, L. Benini, G. De Micheli. *Low-power job Scheduling for Multiple Device*, Proc. of Int. Workshop HW/SW Co-design, pp. 39-43, 2000.
[25] C. Moser, D. Brunelli, L. Thiele, L. Benini. *Real-time scheduling for energy harvesting sensor nodes*, Real-Time Systems, Volume 37, Issue 3, pp. 233-260, 2007.
[26] C. Moser. *Power Management in Energy Harvesting Embedded Systems*, PhD Thesis, ETH Zurich, 2009.
[27] S. Priya, D.-J. Inman. *Energy Harvesting Technologies*, Springer-Verlag, New York (USA), 2009.
[28] V. Raghunathan, S. Ganeriwal, M. Srivastava. *Emerging Techniques for Long Lived Wireless Sensor Networks*, IEEE Communications Magazine, pp. 108-114, 2006.
[29] C. Renner, V. Turau. *CapLibrate: Self-Calibration of an Energy Harvesting Power Supply with Supercapacitors*, Proc. of 23rd Int. Conference on Architecture of Computing Systems, pp. 1-10, 2010.
[30] C. Rusu, R. Melhem, D. Mosse. *Multiversion scheduling in rechargeable energy-aware real-time systems*, Proc. of 15th Euromicro Conference on Real-Time Systems, pp. 95-104, 2003.
[31] M. Spuri. *Analysis of Deadline Scheduled Real-Time Systems*, Technical Report RR-2772, INRIA, Le Chesnay France, 1996.
[32] S. Vestal. *Fixed-priority sensitivity analysis for linear compute models*, IEEE Transactions on Software Engineering, Vol 20, no. 4, pp. 308-317, 1994.
[33] F. Yao, A. Demers, et al. *A Scheduling Model for Reduced CPU Energy*, Proc. of 36th IEEE Symposium on Foundations of Computer Science, pp. 374-382, 1995.

Maryline Chetto received the degree of Docteur de 3ième cycle in control engineering and the degree of Habilitée à Diriger des Recherches in Computer Science from the University of Nantes, France, in 1984 and 1993, respectively. From 1984 to 1985, she held the position of Assistant professor of Computer Science at the University of Rennes 1, while her research was with the Institut de Recherche en Informatique et Systèmes Aléatoires, Rennes. In 1986, she returned to Nantes and is currently a professor with the Institute of Technology of the University of Nantes. She is conducting her research at IRCCyN institute. She has published more than 100 journal articles and conference papers in the area of real-time operating systems. Her current research interests include scheduling and power management for real-time energy harvesting applications.